Computer Graphics
and Multimedia

RWTH AACHEN UNIVERSITY

Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehrstuhl für Informatik 8
Prof. Dr. Leif Kobbelt

---

**Bachelor-Arbeit**

# Efficient Rasterization of Swept Triangle Volumes

Fabian Beckmann
Matrikelnummer: 312112

September 2014

---

Erstgutachter: Prof. Dr. Leif Kobbelt
Zweitgutachter: Prof. Dr. Bastian Leibe

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, September 6, 2014

Fabian Beckmann

# Contents


**1 Introduction** **1**
- 1.1 Motivation . . . 1
- 1.2 Contribution . . . 2
- 1.3 Structure of the Thesis . . . 3

**2 State of the Art** **5**
- 2.1 Data Structure . . . 5
- 2.2 Related Work . . . 6
  - 2.2.1 2D-based Approaches . . . 6
  - 2.2.2 Volume-based Approaches . . . 6
  - 2.2.3 Surface-based Approaches . . . 7

**3 Mathematical Problem Definition** **9**
- 3.1 Sweep Trajectory . . . 9
- 3.2 Swept Volume . . . 10
- 3.3 Swept Volume Envelope . . . 10
  - 3.3.1 Ruled Surfaces . . . 11
  - 3.3.2 Developable Surfaces . . . 11

**4 Subtracting Polyhedrons from the Nail Model** **13**
- 4.1 The Nail Model Data Structure . . . 13
- 4.2 Mesh Rasterization . . . 15
- 4.3 Interval Calculation by Counting . . . 15

**5 Swept Mesh Volume Approximation** **17**
- 5.1 Approximating Swept Triangles . . . 17
- 5.2 Approximating Swept Meshes . . . 18
- 5.3 Approximating With Multiple Trajectories . . . 18
- 5.4 Error of Simple Sweep Steps . . . 19
- 5.5 Error of Non-Simple Sweep Steps . . . 21
- 5.6 Refining Non-Simple Sweep Steps . . . 23

**6 Surface Culling** **25**
6.1 Uniting Polyhedrons . . . . . 25
6.2 Back-facing Volume Culling . . . . . 27
6.3 Bounding Box Culling . . . . . 28
6.3.1 Bounding Box Calculation . . . . . 28
6.3.2 Multi-Step Bounding Box Culling . . . . . 28

**7 Performance Analysis** **31**
7.1 Quality Improvements . . . . . 31
7.2 Test Files . . . . . 32
7.2.1 Torus Flip Testfile . . . . . 32
7.2.2 Tilting Cylinder Testfile . . . . . 32
7.2.3 Swarfing . . . . . 33
7.3 Accuracy . . . . . 33
7.4 Performance . . . . . 34
7.4.1 Single-Cut performance . . . . . 34
7.4.2 Multiple-Cut performance . . . . . 35

**8 Conclusion** **37**

# Chapter 1

# Introduction

## 1.1 Motivation

A common way to produce an arbitrary object is to take a solid piece of material (workpiece) and to cut material away which is not needed. The target shape is usually designed at the computer, in the form of a CAD model. From this model a series of motions is generated, by hand or automatically, which are then performed by specially designed machines.

It is not always easy forecast whether the programmed series of motions will, in fact, create the required shape. Therefore it is necessary to test in advance how big the deviation between the CAD model and the resulting workpiece will be. This could be done by feeding the motion data to the machine to create a test model from a cheap material, but this is a time- and money-consuming process. Therefore, a computer simulation of the material removal process is useful. The simulation shall allow the user to see the removal process in real-time, and it shall also allow measuring the deviance between model and workpiece. Furthermore, it shall check if non-cutting parts of the machine would collide with the workpiece during the cutting process to prevent damage to the machine.

Motion simulation algorithms are classified by the type of machines they are able to simulate. Today, a wide range of machines exist which allow computer-aided manufacturing. It is useful to categorize them by the shape their cutting parts have, and by the range of motion they can perform. In drilling machines a combination of cylinder and cone is sufficient to model them, milling machines are usually modeled by a combination of cylinder, torus and cone. Sawing machines can be modeled by cuboids. To simulate these machines usually specific algorithms are used, which work with implicit or explicit formulas to generate the surfaces. More complex tools and

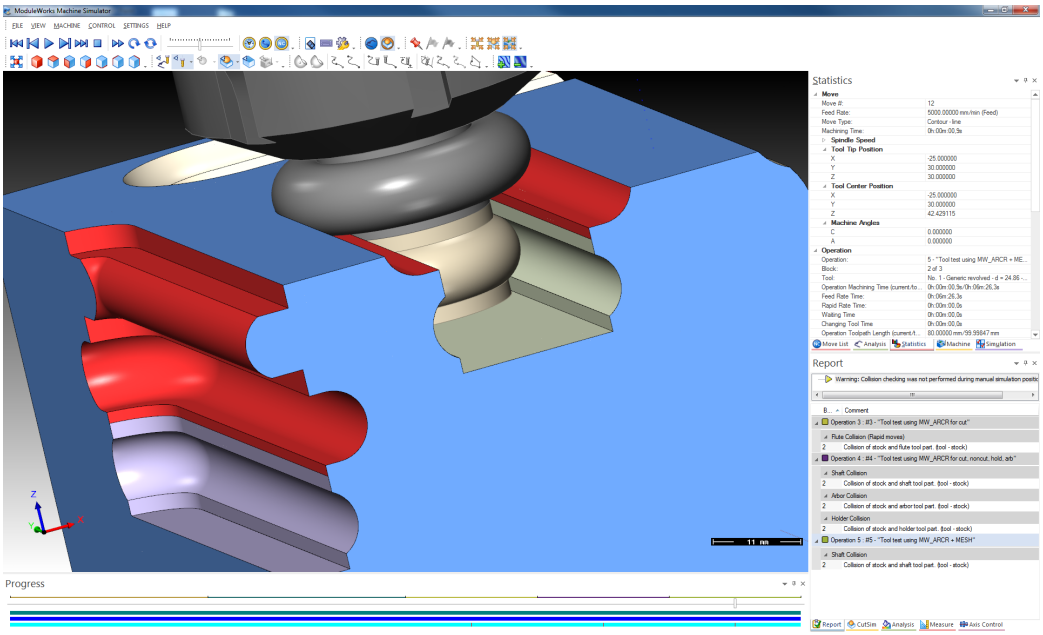



**Figure 1.1:** Example for a CAM simulation software. The volume subtraction is shown on the left, additional statistics on the right side.

non-cutting machine parts are generally modeled by meshes, as their shapes would be too complex to work with if described analytically.

The range of motions machines can perform can be abstracted by their number of axises of freedom. Almost all cutting machines are able to position their tool at a specified location and orientation in space, then they perform a linear move to another position and orientation. A 3-axis machine, for example, is only able to perform translations in all three directions of the coordinate system. A 3+2-axis machine can also change the tool orientation, but only in-between moves, not during them. A 5-axis machine can perform translations and simultaneous rotations during cutting.

## 1.2 Contribution

The algorithm presented in this thesis focuses on 5-axis machining. We investigate, whether and how a data structure called “nail model” can be used to support the simulation process. Nail models are similar to height fields as they sample the volume at the same positions, but instead of storing only the highest point they store all intervals that contain volume and therefore allow the representation of arbitrary volumes, given enough sample points.

In short, the algorithm gets a mesh, a trajectory and a tolerance as input. To approximate the sweep it will calculate a polygonal approximation for each triangle sweep and rasterize it into the nail model. Interior points are then removed from the nail model, such that only points on the envelope of the swept mesh remain. After this, the envelope can be extracted using iso-surface extraction [SLL94].

Furthermore we'll present a method to approximate swept triangle volumes with polyhedra, fulfilling the requirement that the volumes of two adjacent triangles with the same trajectory have to be gap-free and comply to a specified tolerance.

## 1.3 Structure of the Thesis

At first we will look at related work in the area of swept volume computation. Then, we'll look at the mathematical definition of the swept volume. In the following parts we'll see how we can generate partial approximations of the swept mesh volume, and how we can unite them with the nail model, complying with a maximum tolerance. In the last part of this thesis we will see how well the algorithm works for a representative selection of real-world use-cases.

# Chapter 2

# State of the Art

## 2.1 Data Structure

The most common way to store the volume which is left during and after the cutting process is that of CSG[1] trees. A CSG tree allows storing unions, intersections and differences between implicitly defined volumes. After the solid construction the volume can be rendered for example by using a ray-tracing algorithm. This data structure is very cheap to work with if the number of solids is small. However, in cutting simulation the number of solids is usually very high as the tool trajectory is subdivided into many short segments. The CSG data structure doesn't scale linearly because the number of intersections increases exponentially when the number of objects increase. This makes it a bad fit for material removal simulation.

Therefore, at ModuleWorks, a discrete data structure called *nail model* is used, which is basically an extended dexel field which can save more than two intersections per dexel. It needs a huge amount of memory and processing time if the number of object it represents is small, compared to CSG trees. The advantage of the nail model is however, that it is only (about) linearly dependant on the number of objects it stores. Also, intersections between objects are always easy to compute, as this boils down to subtracting an interval from a set of intervals which is very cheap, compared to general solid-solid intersection required in CSG-Trees. The volume can then be reconstructed with a specialized marching cube algorithm, which is not part of this thesis.

As the number of solids created during cutting simulation is usually very high, the time savings gained by the linear time- and space-complexity in practice almost always outweight the greater initial time- and space-requirements.

[1] CSG: Constructive Solid Geometry

## 2.2 Related Work

Swept Volume computation has been extensively researched for more than five decades, as it is the foundation of today's CAD[2] and CAM systems. Different approaches have been successful, which we'll shortly present here.

### 2.2.1 2D-based Approaches

For many use cases it is sufficient to compute a 2D image with depth information of the swept volume. A common approach is to create triangulated representations of all surfaces contributing to the envelope. These can then be rasterized efficiently on GPUs using the Z-Buffer to extract the envelope [HO94]. This approach is very fast, but with the drawback that representation of concave shapes is impossible. Also, 3D reconstruction is difficult for features which are not located parallel to the image plane (see also Figure 4.2). Therefore this approach can only be used in 3-axis machining.

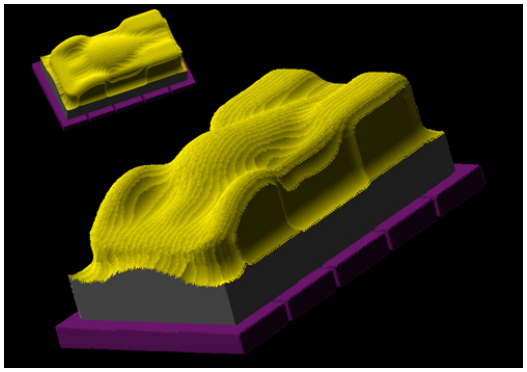


**Figure 2.1:** 3D representation of a 2D depth buffer representation. Image from [HO94].

### 2.2.2 Volume-based Approaches

Swept volume has been investigated using different mathematical techniques, such as Jacobian rank-deficiency conditions [AY97] and Minkowski sums [LKE98]. These approaches solve differential equations based on swept volumes and the move trajectory to find the envelope[3] of the sweep. Therefore,

[2] CAD/CAM means Computer Aided Design and Computer Aided Manufacturing. It describes the process of designing and manufacturing products with the help of computers.

[3] The surface of the swept volume is called envelope.

the mathematical formulation of SV computation is relatively well-understood [WL90].

Methods based entirely onto solving the differential equations of swept volumes are only useful in practice if the solutions can be found in a numerically stable and efficient way. This is why these approaches are only used for very simple solids like cylinders, cones and balls, as well as combinations of these shapes. They are also successfully used for revolved tools[4] [AY06].

Non-cutting machine parts and special-purpose tools can however have arbitrary shapes. Finding solutions to their analytical sweep is usually not feasible. Therefore, algorithms calculating general swept volumes need to work with approximations of the trajectory and the swept solid. A common practice is to use polyhedral approximations.

A simple approach to create an approximation of arbitrary swept solids is presented in [SLL94]. It is based on a 3d-grid, where each grid point stores the distance to the swept volume. To calculate these distances the volume is sampled at multiple points on the trajectory. Each time the minimum distance between the solid and all grid points is updated. The envelope of the swept volume can then be calculated using iso-surface extraction via marching cubes based on a distance function evaluated at each point of the grid. However, this approach is very slow if a small error needs to be guaranteed, as the number of sampled volumes will be very high. Also, the distance function between a point and the solid is often expensive to compute.

### 2.2.3 Surface-based Approaches

In many approaches, the goal is to extract the surface of the swept volume. These algorithms are commonly based on the following three steps:

1. Find all surfaces that contribute to the envelope of the swept volume
2. Compute intersections between all of the surfaces, build a network of surface neighborship, and trim intersecting surfaces
3. Traverse the resulting structure and extract the surface segments that lie on the outer hull

When finding the envelope surfaces, most authors have tried to derive formulas which describe the sweep-generated surfaces explicitly. This however means that they are limited to specific tool-shapes and trajectories, as the algebraic complexity of surface-surface intersection would become too

[4] Revolved tools are defined by a 2d contour, which is revolved around an axis to create a 3d solid.

high otherwise [WL90]. Triangulated approximations of these surfaces have the advantage that they are easier to handle, and intersections are cheap to compute. Also, there are successful approaches which approximate triangulated envelope surfaces for polyhedrons performing arbitrary continuous trajectories [Kim+04; FMM86].

Instead of traversing the outer hull of the generated surfaces, [Kim+04] proposes envelope extraction based on the marching cubes algorithm. Marching cubes is a simple way to discover all surfaces which are reachable from infinity. After marking those surfaces, all interior surfaces can be discarded. All of the approaches based on unstructured surface-primitives have the problem that interior cavities in the swept volume are filled, as they are not reachable from outside and it's not easy to solve this problem based on the partial surfaces only. The algorithm presented in this thesis doesn't have that problem and is able to accurately represent cavities.

# Chapter 3

# Mathematical Problem Definition

## 3.1 Sweep Trajectory

The algorithm is given a polyhedron $\Gamma$ bounded by triangles $\Gamma_1 \dots \Gamma_n$. Each point $p$ of the polyhedron is swept along a trajectory defined by 3D points and orientation-defining quaternions:

$$\{(p_1, q_1), (p_2, q_2), \dots, (p_n, q_n)\}.$$

Between each pair $(p_i, q_i)$ and $(p_{i+1}, q_{i+1})$ the motion is calculated using Slerp[1] [Sho85]. This results in a continuous trajectory $\tau(t)$ as Slerp itself is continuous, and the position as well as the orientation between adjacent Slerp segments is the same.

For analyical purposes it is helpful to use a simplified representation of the Slerp trajectory. By using matrices instead of quaternions and an appropriate coordinate transformation we can express the interpolation between two coordinates as such:

$$T(\vec{p}, t) = R_{\vec{a}}(t \cdot \theta) \cdot \vec{p} + t \cdot \vec{d}.$$

This formula describes a linear, simultaneous translation and rotation. $t$ is a parameter in $[0, 1]$. $R_a$ is the rotation matrix with axis $\vec{a}$, $\theta$ the sweep angle and $\vec{d}$ the translation vector.

We can determine the position of a polyhedron $\Gamma$ by transforming it's points according to the translation and rotation defined by $\tau(t)$. We denote this as $\Gamma^{\tau(t)}$.

[1] Slerp: Spherical Linear Interpolation

## 3.2 Swept Volume

The swept volume of a polyhedron $\Gamma$ is defined as the union of the volumes enclosed by the polyhedron in all possible positions on the trajectory:

$$SV := \bigcup_{t \in [0,1]} \Gamma^{\tau(t)}$$

This definition of the swept volume is not sufficient for most applications. Usually, it is necessary to generate the surface of the swept volume, often called *envelope*. The surface, for example, allows easy visualization of the volume using rasterization as well as distance computation to a reference CAD model.

## 3.3 Swept Volume Envelope

Given an implicitly defined surface of the swept object

$$f(x, y, z) = 0$$

and it's trajectory, it is easy to derive a family of surfaces, each of them describing the envelope at a given point in time $t$:

$$F(x, y, z, t) = 0.$$

As shown in [MS89] the envelope of this surface family, which encloses all of the family members, can be determined by simultaneously satisfying

$$F(x, y, z, t) = 0, \qquad \frac{\partial F(x, y, z, t)}{\partial t} = 0.$$

The implicit form of the envelope surface can then be reconstructed by eliminating $t$, giving

$$e(x, y, z) = 0.$$

Instead of computing the swept volume for general shapes, we want to concentrate on triangles. The swept volume envelope of a flat polygon $f$ has been extensively researched in [WL90]. It has been shown that it consists of:

1. The faces $f^{\tau(0)}$ and $f^{\tau(1)}$
2. Three ruled surfaces generated by the edges
3. A developable surface generated at non-edge points where $f^{\tau(t)}$ becomes grazing (i.e. lies on the envelope)

Some of these surfaces lie in the interior of the swept volume and thus need to be clipped to get a valid representation of the envelope.

### 3.3.1 Ruled Surfaces

Given an edge $\vec{l}$ with distance vector $\vec{d}$ from the origin and a Slerp trajectory $T$ we can calculate the ruled surface explicitly with this formula:

$$x(u,v) = u \cdot \vec{t} + R_a(u \cdot \theta) \cdot (v \cdot \vec{l} + \vec{d})$$

### 3.3.2 Developable Surfaces

When a face moves along a trajectory $\tau$ it's face may come in contact with material which is not removed yet. The resulting surface, as shown in [WL90], is placed where $f^{\tau(t)}$ and $f^{\tau(t+\varepsilon)}$ intersect. [WL90] also showed that this surface is always developable.

We can parametrically represent a moving plane $p(u,v,t)$ as:

$$p(u,v,t) = q(t) + u \cdot r_1(t) + v \cdot r_2(t)$$

where $q(t)$ is the origin of $p(u,v,t)$ and $r_1(t)$ and $r_2(t)$ are two linearly independent vectors spanning the plane. This plane will become grazing at all points where the move direction is parallel to the plane.

Using envelope theory we can find the solution by solving $det(J(p(u,v,t)) = 0$ for u. [WL90]

The developable surface $d(t,v)$ is therefore:

$$d(t,v) = b(t) + v \cdot \delta(t)$$

where

$$b(t) = q(t) - r_1(t) \cdot \frac{q'(t) \cdot r_1(t) \times r_2(t)}{r'(t) \cdot r_1(t) \times r_2(t)}$$

$$\delta(t) = r_2(t) - r_1(t) \cdot \frac{r'(t) \cdot r_1(t) \times r_2(t)}{r'(t) \cdot r_1(t) \times r_2(t)}$$

This derivation is only valid if $r'(t) \cdot r_1(t) \times r_2(t) \neq 0$. Otherwise we can derive a similar equation by solving for $v$.

The swept triangle will generate such a developable surface at it's envelope. The edges $e_1$ and $e_2$ define the plane direction vectors $r_1$ and $r_2$. The developable surface may lie partly or entirely outside the triangle, therefore it needs to be clipped to the triangle bounds.

# Chapter 4

# Subtracting Polyhedrons from the Nail Model

## 4.1 The Nail Model Data Structure

To represent the material volume a special data structure called *nail model* is used. The nail locations are referenced by a two-dimensional grid in the xy-plane of the nail model coordinate system. Each of the nails can be referenced by an integer pair $(I_x, I_y)$. As the nails are uniformly distributed, the nail-space coordinate can be calculated as $(I_x\omega_x, I_y\omega_y)$, given the scaling parameters $\omega_x$ and $\omega_y$.

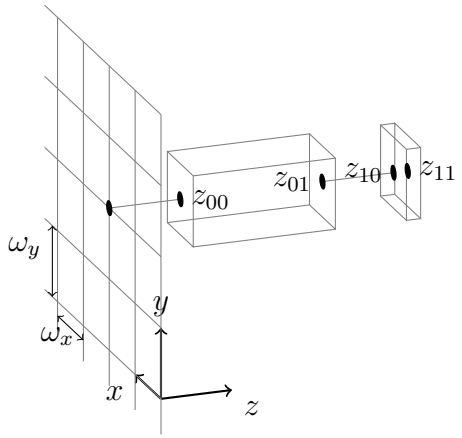
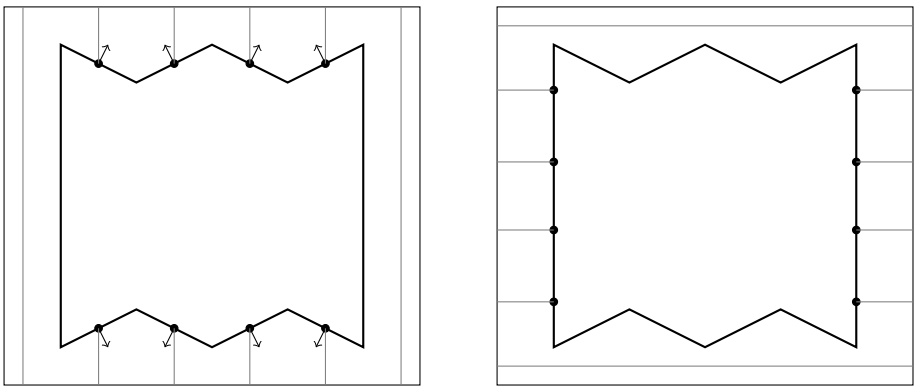



**Figure 4.1:** Nail Model coordinate system

Each nail describes a set of cuboid solids, extending along the z-axis of the nail coordinate system. The x- and y-dimension of each nail are fixed, and the z-depths are determined by a set of pairs $\{(z_{00}, z_{01}), (z_{10}, z_{11}), \ldots (z_{n0}, z_{n1})\}$.

The accuracy of the nail model is determined by the orientation of the xy-plane and by the scaling parameters $\omega_x$ and $\omega_y$. For maximum accuracy the area where most features are should be parallel to the xy-plane, as seen in Figure 4.2. In general, there are three methods to guarantee this: letting the user choose the transformation, automatic estimation, and using multiple nail models. For our purposes only the last option is feasible: we usually want to perform multiple cuts on the workpiece, which may have arbitrary entry directions and swept volume shapes. The cuts are also not known in advance, but they are fed to the algorithm on-the-fly, while the intermediate results need to be shown. Therefore we'll use one nail model for xy-, xz- and yz-plane each. This ensures sufficient accuracy for arbitrary locations of important features.



(**a**) shape is represented well (**b**) shape is badly represented

**Figure 4.2:** Importance of the rasterization direction

The size of $\omega_x$ and $\omega_y$ has to be set to the user-specified tolerance $E$. This ensures that the maximum error due to the representation is $E$, while the medium error will be $\frac{E}{2}$.

When calculating the intersection of a polyhedron with the nail model the intersection points between the polyhedron and each nail are computed and the intervals between the intersection points are subtracted from the nail. Additionally, surface normals are generated.

Combining intersection intervals and normal information for all three nail fields with a specialized marching-cube algorithm allows an efficient triangulation of the envelope. In [Ben+97] an algorithm was presented which solves this problem for dexel fields[1]. With slight modification this method also works for the nail model, which is implemented in the ModuleWorks code base.

[1]Dexel fields are similar to the nail model, but they allow only one z-volume interval.

## 4.2 Mesh Rasterization

The meshes generated in the sweep approximation will all be single component, closed, triangular, outward-oriented, not-self-intersecting manifolds, hereinafter referred to as *well-formed*. Instead of subtracting each of these meshes individually, we will use an intermediate representation and rasterize them all at once. The intermediate representation doesn't collect intervals, but it just stores a set of unstructured intersection points including the surface normals.

The intermediate model is called *consistent* if it fulfills the following properties:

**Definition 1.** Consistency of nails

1. Each nail has to have an even number of points
2. The normal orientation has to alternate
3. The normal of the first point has to be outward-oriented

Consistency guarantees that we can extract the intervals containing volume: Each volume interval is delimited by two points with normals pointing outside of the interval.

Rasterization of a simple mesh will always yield a consistent nail model, as any line passing through it will alternately enter and leave the interior of the mesh. Therefore the number of intersection points is even, and the normal orientation alternates. In view of numerical instabilities it is not immediately clear that this property holds: triangles sharing an edge could be rasterized both or not at all. However, this problem can simply be solved by adapting to consistency rules, which is implemented by most modern rasterizers [Mic14].

## 4.3 Interval Calculation by Counting

When rasterizing multiple simple meshes into the nail model it will not necessarily be consistent anymore. However, we can restore consistency by simply iterating over the points: Whenever we find a normal pointing toward the start we increase a counter by one, when it points away, we decrease it. The counter therefore counts the number of volumes we currently are in. During the iteration we then can simply remove all points found when already being inside another hull.

With this technique we can build a simple, stable algorithm that approximates the sweep subtraction. We just need a way to generate one or multiple

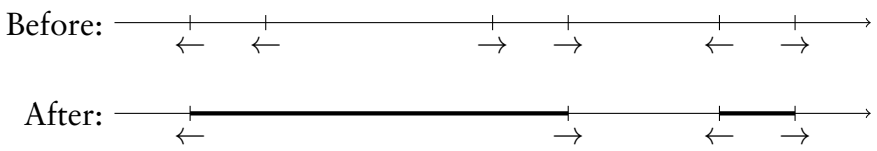



**Figure 4.3:** An intermediate nail becomes a real nail by culling interior points.

simple meshes with correct normal information. In the following sections we'll describe how build the approximation meshes.

# Chapter 5

# Swept Mesh Volume Approximation

Exact rasterization of the swept meshes is computationally very complex. Each of the boundary surface is a multidimensional polynomial, therefore rasterization is expensive. Computing the intersections of these surfaces to generate the envelope is even more challenging. As speed is very important in practice, and accuracy is only important in known and specified bounds, we will use an approximated representation of the swept volume. Therefore we will generate a set of polyhedrons that deviates by no more than a distance of $E$.

In this chapter we will first focus on trajectories which are defined by just two coordinates, i.e. one Slerp interpolation. We will then see how to extend this method to an arbitrary number of coordinates.

## 5.1 Approximating Swept Triangles

A simple way to approximate the swept volume between two faces $f^{\tau(t_0)}$ and $f^{\tau(t_0+d)}$ is to use the smallest convex hull containing these triangles. As our trajectory is always continuous it is guaranteed that the convex hull will always be in tolerance if $d$ is chosen sufficiently small. This means, that the approximation converges towards the optimum.

To approximate the sweep of a single polygon over the entire trajectory from $f^{\tau(0)}$ to $f^{\tau(1)}$ we will use multiple polyhedrons: the triangle is sampled in multiple positions of the move, so we yield multiple triangles $T_1, ..., T_n$. Between each triangle pair $T_i, T_{i+1}$ we approximate the sweep by calculating the convex hull of the six involved vertices. We will later see how to determine the number of interpolation steps $n$ necessary to approximate the sweep in

tolerance.

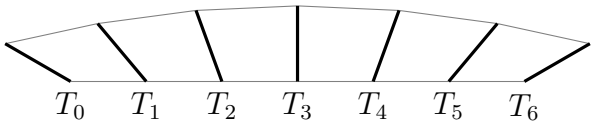



**Figure 5.1:** Triangles in 2d projection, with convex hulls in gray

## 5.2 Approximating Swept Meshes

Based on the swept triangle approximation we want to approximate a swept polyhedron approximation. When constructing the swept triangle approximations two properties are important to keep in mind: the polyhedrons need to be in tolerance, and they need to be gap-free. Tolerance is self-explanatory: when all of the swept triangle approximations are in tolerance, then the union will approximate the mesh in tolerance also. It would, however, be possible to generate polyhedrons for adjacent triangles that respect the tolerance $E$, but have gaps between them. This must be prevented as it is a very noticable error to the user, and it would also mean that the union would most likely not be in tolerance to the analytical mesh sweep anymore.

Guaranteeing that the set of swept triangle approximations is gap-free is rather easy: if we chose the same trajectory sampling positions for each triangle, then using convex hulls guarantees us that hulls of adjacent triangles don't have gaps between them. This will of course over-tesselate triangles which are close to the rotation axis if we want the triangles far away to stay in tolerance, but it guarantees correctness. For simplicity reasons we'll also use uniform sampling, although chosing non-uniform step sizes would be easy to do.

The set of approximation polyhedrons can easily be united with the nail model.

## 5.3 Approximating With Multiple Trajectories

We have previously seen how to compute the swept volume between two coordinates $(p_i, q_i)$ and $(p_{i+1}, q_{i+1})$. To compute the swept volume for trajectory $T$ we have two options: we can either generate all the polyhedrons first and then rasterize them into the nail model to unite them, or we can rasterize each of them into an individual nail model and then unite the multiple nail models. A mixture between those methods, i.e. rasterizing the volume between multiple

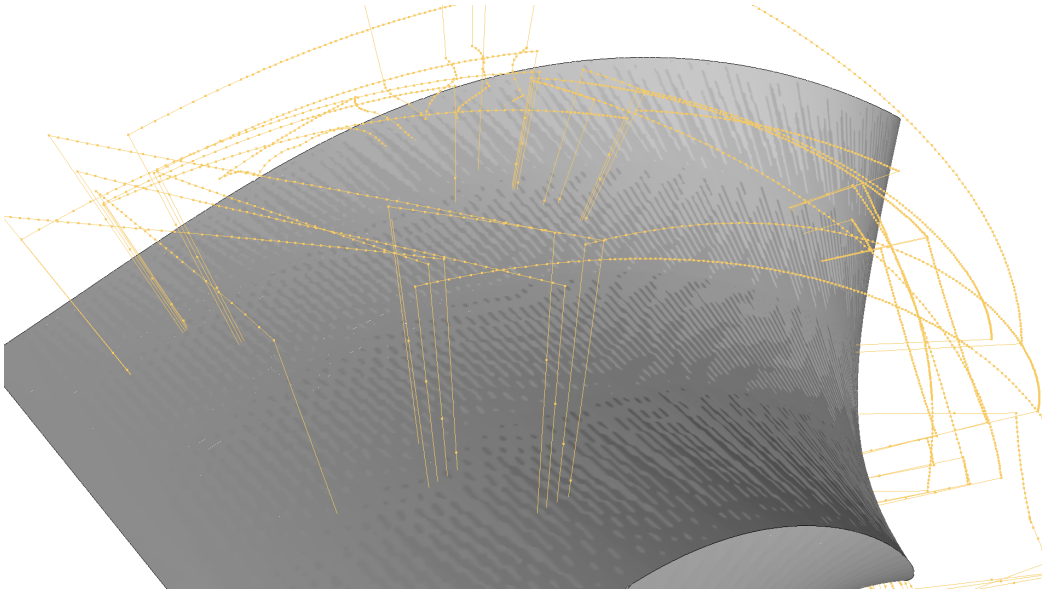


**Figure 5.2:** The complete trajectory is defined by multiple sample points.

adjacent coordinates into the same nail model, and then uniting those, has proven to be the best option regarding memory and time efficiency.

## 5.4 Error of Simple Sweep Steps

Given a maximum error $E$ we can calculate the necessary number of interpolation steps $n$. $E$ describes the maximum allowed distance between our approximation and the perfect sweep. To do so, we first introduce a new property for the sweep step approximation: Simplicity.

**Definition 2** (Simple Sweep Step)**.** Let $H \subset \mathbb{R}^3$ be the halfspace defined by the plane of $f^{\tau(t_0)}$ and it's normal vector. The sweep approximation step between two triangle positions $f^{\tau(t_0)}$ and $f^{\tau(t_1)}$ is called *simple*, if the motion vectors of each point are directed into $H$ at any time during the motion.

The convex hull of a simple sweep step is structured in such a way, that $f^{\tau(t_0)}$ and $f^{\tau(t_1)}$ lie on the convex hull. They are connected by three triangulated quads. This makes error approximation fairly straightforward.

When approximating the error of simple sweeps we have to take the different characteristics of the surfaces into account. The start and end triangle lie on the convex hull, therefore the error of these surfaces is guaranteed to

be zero. The other six triangles approximate the ruled surfaces generated by the swept edges.

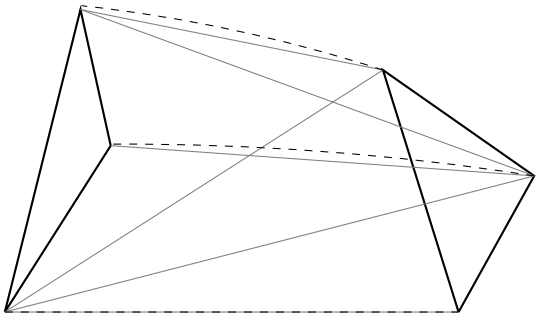


**Figure 5.3:** Simple Sweep Step. Dashed: analytical swept vertices; Gray: convex hull triangles

The error of linear surface approximation has been extensively researched in the past. In [FMM86] it has been shown how to approximate the error between a $C2$[1]-surface and a piecewise linear approximation based on the second-order derivative. We can use these results to get a lower bound for the number of approximation steps we need to satisfy the error bound $E$.

The number of approximation steps needed can be calculated using this formula, as shown in [FMM86]:

$$n \geq \frac{1}{8E}(M_2 + \sqrt{M_2^2 + 8E \cdot M_1})$$

where $M_1$ and $M_2$ are defined over the domain $T = [0,1] \times [0,1]$ as:

$$M_1 = \sup_{(u,v)\in T} \left\| \frac{\delta^2 f(u,v)}{\delta u^2} \right\|$$

$$M_2 = \sup_{(u,v)\in T} \left\| \frac{\delta^2 f(u,v)}{\delta u \delta v} \right\|$$

When we apply these results to the ruled edge surface formula from

[1] $C2$ means that the surface function, and it's first- and second-order derivative need to be continuous.

Section 3.3.1 we get the following values:

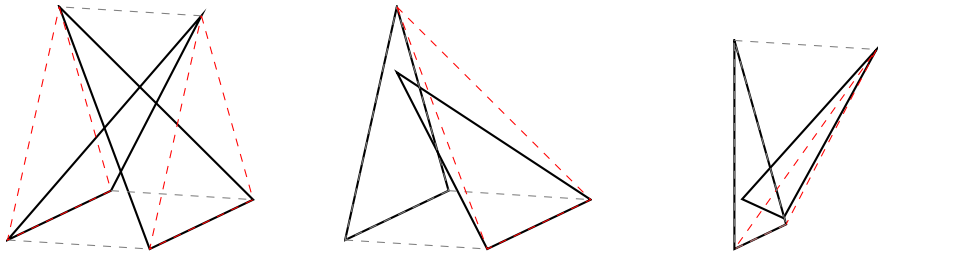
$$\begin{aligned} M_1 &= \sup_{(u,v)\in T} \left\{ \left\| \begin{pmatrix} -(d_x + l_x)\theta^2 \cos(u\theta) + (d_y + l_y)\theta^2 \sin(u\theta) \\ -(d_y + l_y)\theta^2 \cos(u\theta) + (d_x + l_x)\theta^2 \sin(u\theta) \end{pmatrix} \right\| \right\} \\ &\leq \theta^2 \cdot \max\{|d_x|, |d_y|, |d_x + l_x|, |d_y + l_y|\} \end{aligned}$$

$$\begin{aligned} M_2 &= \sup_{(u,v)\in T} \left\{ \left\| \begin{pmatrix} -l_y\theta \cos(u\theta) - l_x\theta \sin(u\theta) \\ l_x\theta \cos(u\theta) - l_y\theta \sin(u\theta) \end{pmatrix} \right\| \right\} \\ &\leq \theta \cdot \max\{|l_x|, |l_y|\} \end{aligned}$$

By calculating $n$ for each edge in the input mesh we can determine the minimum number of steps that is necessary such that all generated edge sweeps, and therefore all simple sweep steps are in tolerance.

## 5.5 Error of Non-Simple Sweep Steps

For non-simple sweep steps the error computation is quite a bit more complex. The convex hull of two equal triangles at different positions in space consists of either 8 (if $T_i$ and $T_{i+1}$ intersect), 6 (if one of the vertices lies inside the hull) or 4 (if two of the vertices lie inside the hull) triangles. See Figure 5.4 for examples.



(a) intersecting triangles (b) one interior vertex (c) two interior vertices

**Figure 5.4:** The three kinds of convex hulls in complex cases. Red marks the problematic triangles.

We can classify the triangles of the bounding box into two categories:

1. Triangles that share a common vertex in start and end triangle

2. Triangles that cover all three vertices in start and end triangle.

We know that faces of the first kind are in tolerance, as their error has been limited when determining the number of sweep steps. These triangles are part of a triangulation of the edge sweep quad. Therefore we only need to care about the error of the second kind of triangles. We'll call these the problematic triangles, and there is either one or two of those triangles in a convex hull of a non-simple sweep step.

Calculating the error of a triangle to the analytical volume is very expensive computationally. To compute the exact error, we would have to calculate the difference between each convex hull triangle and the piecewise surfaces described in Chapter 3.2. Therefore we'll select a set of carefully chosen sample points from the analytical surface, and measure the distance to the problematic triangles.

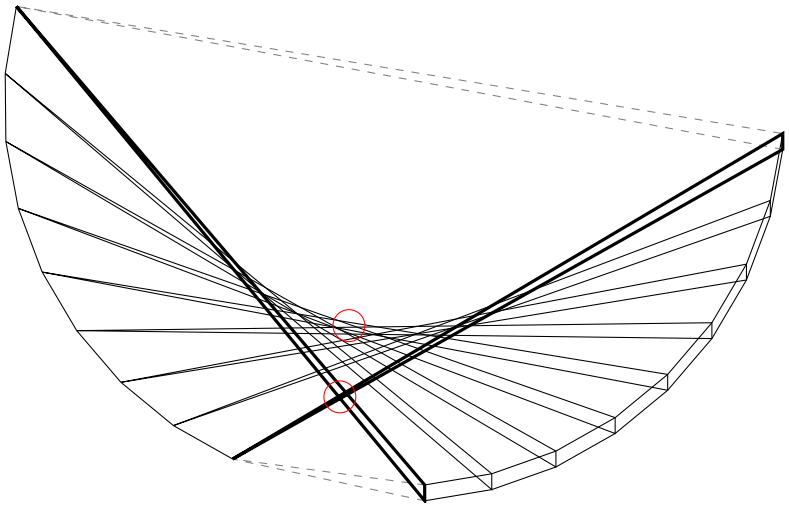


**Figure 5.5:** Rotating triangle with developable surface on top, generated by the intersecting triangles. The bottom surface is built only by the intersecting start and end triangles. Red circles mark the sample points where the error between real sweep and the two problematic hull triangles is measured. Dashed lines mark the problematic hull triangles.

We know that the problematic surfaces either approximate the developable surface, or the surface that is is generated by the intersecting start- and end triangles (Figure 5.5).

The developable surface can be calculated as described in Section 3.3.2, but we can calculate it in an easier way: the surface is generated by the intersection of triangles $f^{\tau(t)}$ and $f^{\tau(t+\varepsilon)}$, with $\varepsilon$ chosen sufficiently small. Therefore, instead of calculating the developable surface line at $t$, which is quite inefficient, we can calculate the intersection between these two triangles.

To determine which sample point is the best for the developable surface we'll use the following heuristic: as the developable surface is generated by a function of rotation (and translation), the maximum deviation between the hull triangle and the real surface can be expected to be in the middle of the developable surface with respect to the generating parameter $t$ (see Section 3.3.2). Calculating the distance between the developable surface line at $\frac{t}{2}$ and the hull triangle gives a good approximation of the error.

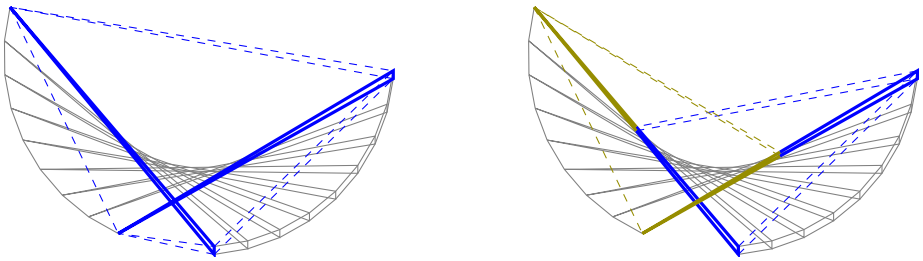
The error between the intersecting start and end triangle and the convex hull triangle covering this surface is always maximal at the intersection line and therefore easy to determine.

This gives us a good approximation of the overall error, but we might unfortunately over- and under-estimate the error between developable surface and the problematic triangle. Therefore we're not able to guarantee to stay below the tolerance. The advantage of this error estimation method is however, that it is very quick and good enough to stay in tolerance for most inputs which were tested (see Section 7.4).

## 5.6 Refining Non-Simple Sweep Steps

The number of required steps isn't easily determined in advance. Also, complex cases sometimes need much more steps than the simple cases (which usually are more common). Adapting the number of steps to the number required for the worst complex case would mean, that the number of triangles needed would increase immensely. Therefore we'll subdivide the problematic traingles into four smaller ones, which also decreases the error. It is easy to see why this decreases the error: if splitting is done often enough, we'll end up with a high number of very tiny triangles (essentially points). The error of swept edges (and therefore all swept points) has been limited in the previous step and therefore is small enough.

In few cases it will not be possible or feasible to split the triangle often enough to stay below the maximum tolerance. This happens, when we're splitting a triangle which has an edge with an approximation error very close to the maximum tolerance, and when this edge lies inside the convex hull. Then we would have to split infinite times. We're not willing to do that, and we also don't want to resample everything again with a higher number of sample steps, therefore the error guarantee is weak; however, in our tests (Section 7.4) the error guarantee has never been violated without careful construction of this special case.



(**a**) Before split (**b**) After split

**Figure 5.6:** The error of problematic triangles decreases after splitting the triangle and uniting the individual convex hulls.

# Chapter 6

# Surface Culling

The most significant measure that influences performance is the number of triangles that need to be rasterized. This number has impact in multiple areas: More triangles means more time spent in the rasterizer, more memory usage in the intermediate nail model, and more time spent when extracting the final nail model.

To reduce the number of surfaces, we will try to eliminate triangles before rasterizing such that the final nail model won't change. This will not be entirely possible, otherwise we could construct the swept mesh envelope without using the nail model at all. However, we can safely remove some of the triangles safely if this unites two polyhedrons into one, and we can leave polyhedrons out which are known to lie entirely inside the swept volume.

## 6.1 Uniting Polyhedrons

Arbitrary polyhedrons can be united with help of the nail model, as shown in Chapter 3.2. To do so, all triangles need to be rasterized into the nail model, which is a quite expensive operation. We can often unite approximation polyhedrons without needing to rasterize them, which saves time.

When selecting candidates for polyhedrons which can be united, we don't want to re-triangulate anything, as this would mean that the number of triangles would increase or that the volume changes. Instead, we will take a rather conservative approach: if two polyhedrons share the same triangle exactly, we can leave it out without changing the result. Also, if two convex hulls share the same four points, we can easily throw the interior triangles away.

Two adjacent convex hulls of simple sweeps can be united without much effort. If they are generated by the same triangle they share a common triangle

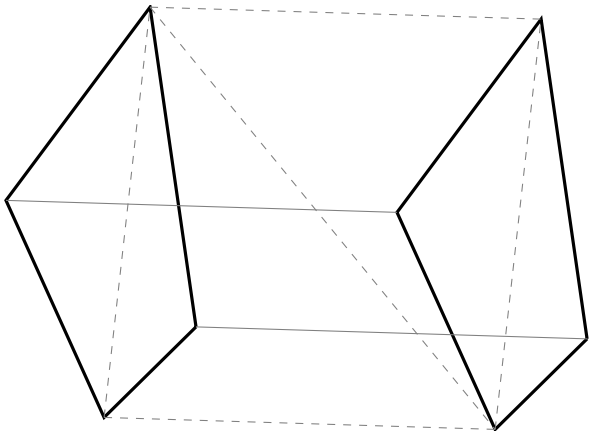


**Figure 6.1:** Two adjacent simple sweeps. The four interior triangles (dashed, two for each sweep step) can be removed.

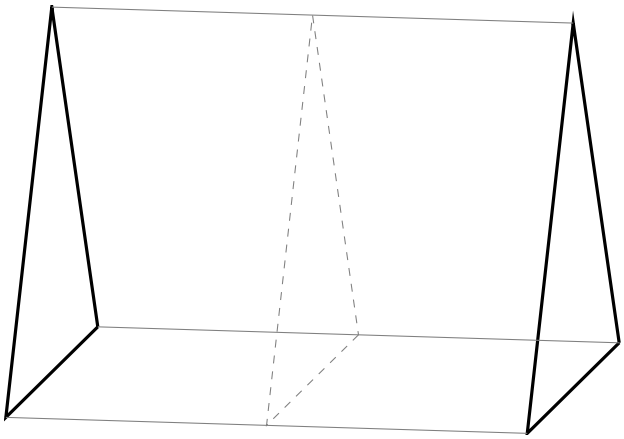


**Figure 6.2:** Two consecutive simple sweeps. The two interior triangles (dashed, one for each sweep step) can be removed. Note: for simplicity reasons the triangulation of the connecting quads is not shown here.

which can be removed in both convex hulls (Figure 6.2). Otherwise they share a common quad, which can be left out, even if the quad is triangulated differently (Figure 6.1).

To find all of the simple triangle hulls that can be united a flood-fill algorithm is used. In each of the sampling steps all adjacent triangles are checked for simplicity, and if both of them are simple they are added to a *patch*. Due to the nature of flood-fill all of those patches are maximal in the end, i.e. no patch has an adjacent simple triangle. When generating the sweep triangles for a sampling step only the exterior triangles of a patch are generated, as all of the interior faces can be left out. This is a generalization

of Figure 6.1, where the patch size is only two.

Furthermore, the generating triangles between sampling steps can be removed, if the consecutive sweep steps are simple. This can be checked on a triangle-by-triangle basis and is not dependant on adjacent triangle properties. An example of this is seen in Figure 6.2.

## 6.2 Back-facing Volume Culling

When sweeping a mesh the front-facing triangles will cut volume away, which is then swept over again by the back-facing triangles. This is obviously unnecessary, therefore we can ignore the back-facing triangles, as we can guarantee that it doesn't affect the output. The property only holds if the mesh in starting position doesn't contain any volume. We can ensure this by checking if the mesh lies outside the nail model bounding box, and if it doesn't, by removing it's volume in a preprocessing step. The mesh volume can be subtracted the same way sweep steps are subtracted, as long as the mesh is *well-formed*.

To determine whether a triangle sweep is back-facing we need to consider the move vectors of each point of the triangle at any position on the trajectory. Only if the vector product between each of these vectors and the triangle normal is negative or zero it can be guaranteed that the triangle will never be on the sweep envelope, with the exception of the start position. It is easy to see why: a triangle that is pointing away from the move direction will necessarily be in the shadow of other triangles, as the swept mesh is always closed and the trajectory is continuous.

The move vectors of all triangle points given a fixed point on the trajectory can always be interpolated linearly with the help of barycentric coordinates. It is therefore sufficient to calculate the move vectors at the triangle vertices and determine the normal cone by linearly interpolating between those three vectors.

As we don't calculate the exact sweep but a linear approximation, determining the vertex move vectors is relatively easy: we can just calculate the distance vectors between the vertices in start and end position.

With these three vertex move vectors we can determine whether a triangle sweep step is backfacing by testing if the vector product between each of them and the triangle normal is negative.

## 6.3 Bounding Box Culling

### 6.3.1 Bounding Box Calculation

For the purpose of Material Removal Simulation calculating the entire swept volume is usually not necessary if we can guarantee that the parts we don't compute don't have an effect on the workpiece. A swept volume has no impact if it lies entirely in empty parts of the nail model, or if the swept volume lies outside the bounds of the nail model.

We can approximate the swept volume using a bounding box. This allows for cheaper tests against the nail model.

A bounding box of a swept volume is defined by two points $\langle p_0, p_1 \rangle$, where $p_0 = (x_{min}, y_{min}, z_{min})$ and $p_1 = (x_{max}, y_{max}, z_{max})$, so that any point $(x, y, z)$ in the volume satisfies

$$\begin{aligned} x_{min} \leq x \leq x_{max}, \\ y_{min} \leq y \leq y_{max}, \\ z_{min} \leq z \leq z_{max}. \end{aligned}$$

In Section 5.4 we have seen how to determine the number of interpolation steps necessary to approximate a swept triangle volume up to a given tolerance $E$. When solving the resulting formula for $E$ we yield an approximation of the error depending on the number of interpolation steps:

$$E \leq \frac{1}{8} \left( \frac{1}{n^2} M_1 + \frac{2}{n} M_2 \right).$$

A bounding box for the exact swept volume can be calculated by generating a small number of interpolation steps. The vertices for each interpolated triangle are united to the tightest bounding box $\langle p_0, p_1 \rangle$ that encapsulates all of them. We can then calculate the approximation error $E$ for the chosen number of steps. It is easy to see that the bounding box $\langle p_0 - (E, E, E), p_1 + (E, E, E) \rangle$ is guaranteed to contain the volume.

When calculating the bounding box of all triangle bounding boxes we yield a bounding box for the entire swept mesh.

### 6.3.2 Multi-Step Bounding Box Culling

The nail model, which represents the final volume, is limited in size. This means, that we can specify a bounding box $B_N$ which describes the dimension of the nail model in world space coordinates. We can save a lot of time if we know that the entire swept volume lies outside $B_N$, as we don't have to

calculate the in-tolerance volume representation (i.e. the steps described in Chapter 5). To test for this case we calculate a bounding box for the entire swept mesh, using one interpolation step.

Additionally, the bounding box $B_T$ for each triangle sweep step can be tested against $B_N$. The test is done in a multi-step process. At first the bounding box of the nail model $B_N$ is compared to $B_T$: if they don't intersect the triangle doesn't need to be rasterized. If the two bounding boxes do intersect we check if the intersecting part of the bounding box lies entirely in empty parts of the nail model. To do this efficiently, we use *nail groups*.

Nail groups are implemented as an Octree [Mea80]. Each cell of this tree is annotated with the information whether the contained area is empty. This is more efficient than checking each nail individually for each bounding box check. The nail group cells can be subdivided finer depending on whether they are questioned often until a pre-defined depth.

# Chapter 7

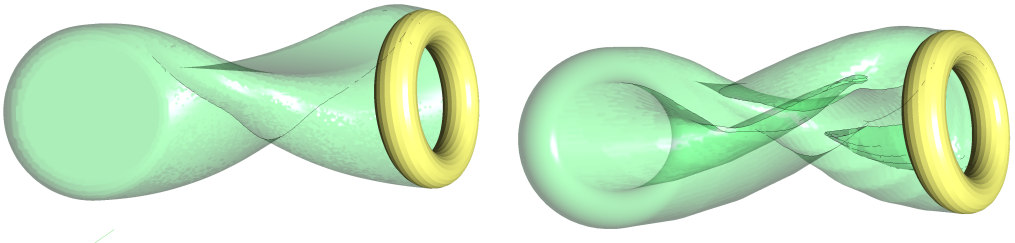
# Performance Analysis

The algorithm's main purpose is that of 3d tool simulation. The chosen test files cover a wide area of applications in the area of cutting simulation and are therefore representative for the real-world usage of this algorithm.

## 7.1 Quality Improvements

The algorithms for 5-axis swept polyhedron reduction previously implemented at ModuleWorks were implemented using a quite naive approach. The trajectory was approximated by a small linear motions. Between each of these motions the rotation changed, thus the solids were subtracted from the nail model. After this, an iso-surface extraction algorithm was used, similar to the one described in [SLL94]. In general the number of interpolation steps had to be chosen very high. Additionally, it was unable to handle cavities.

The new algorithm presented here allows cavities and is much faster than the previous approach.

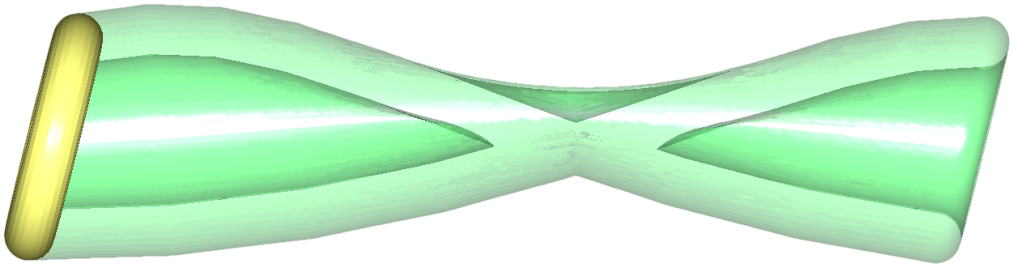


(**a**) Old algorithm (27s)

(**b**) New algorithm (0.56s)

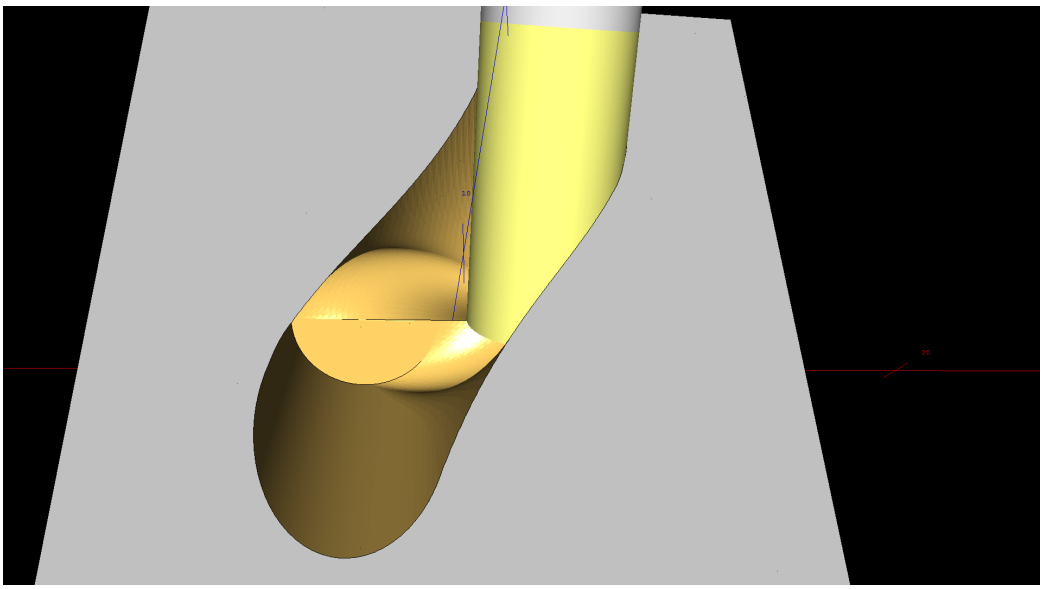
## 7.2 Test Files

### 7.2.1 Torus Flip Testfile



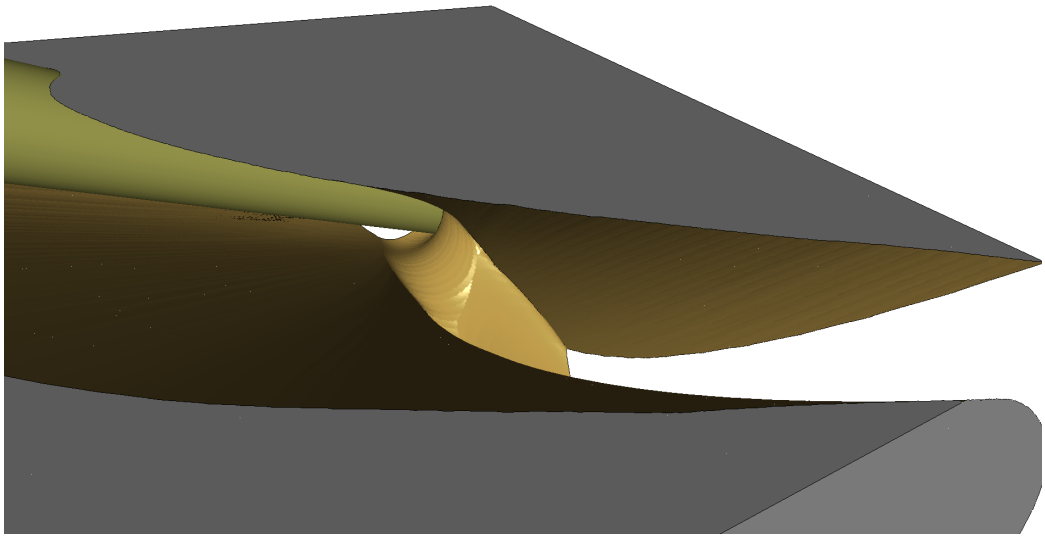
This test file consists of a torus, that is swept to the left side and flipped during this motion. As the rotation axis is orthogonal to the move direction most of the sweep steps are simple. Therefore, the surface culling removes most of the inner faces.

### 7.2.2 Tilting Cylinder Testfile



This test file describes a cylinder that is moved from left to right, with the rotation axis being equal to the move direction. It is tricky, as most of the triangle steps need to be generated from two intersecting triangles.

### 7.2.3 Swarfing



This test file is similar to one seen in practice during a swarfing process. In swarfing big chunks of volume are cut away, apart from a small area which holds the two pieces together to prevent the removed part from uncontrollably flying away and potentially doing damage to equipment or operating personnel.

The test file is described as a series of 80 very short cuts.

## 7.3 Accuracy

Accuracy is best measured when performing a single, long cut because then the algorithm decides how many interpolation steps need to be introduced. When simulating a series of very small cuts the trajectory is often over-sampled anyway, therefore the error is smaller than the tolerance even if every cut is performed with just one interpolation step.

| | torus flip | tilting cylinder |
|---|---|---|
| no optimizations | 753552 tris | 222904 tris |
| backface culling | 364559 tris | 47874 tris |
| all optimizations | 42152 tris | 39008 tris |
| simple sweep steps | 92% | 25% |
| maximum error | 0.5 | 0.88 |

The test show that backface culling eliminates usually more than the possibly expected 50% of the triangles. This is because backfacing triangles

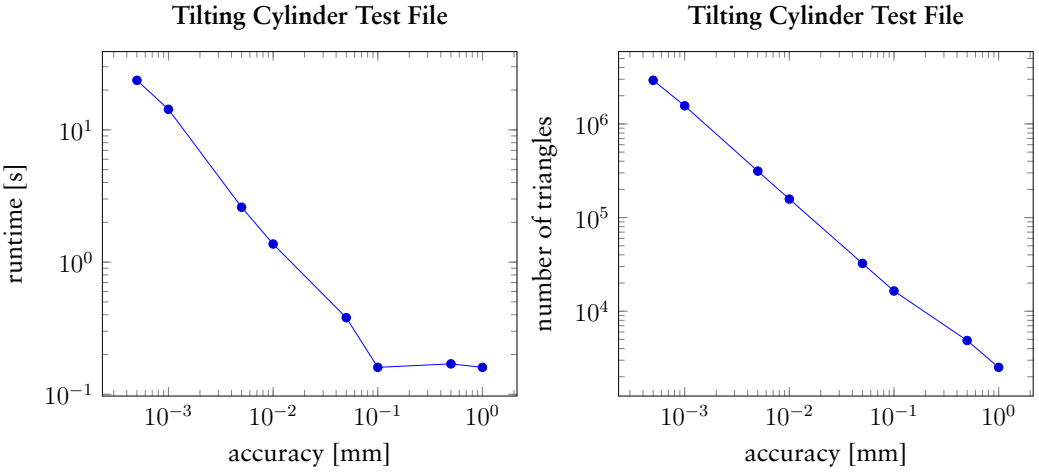
don't need to be taken into account during the determination of the required number of interpolation steps. Less steps mean that way less triangles need to be created.

Duplicate Surface Culling only has much of an effect if many of the sweep steps are simple. Only then many of them are adjacent which causes most interior surfaces to be removed. But in all of the test files the Duplicate Surface Culling step saved more time than it costed.

All test files have been rasterized with a requested tolerance of 1. It can be seen that the error is always below this, but fairly close to it. When the error is way lower than the requested tolerance this is because the surfaces with maximum error are cut away by other triangle sweeps.

## 7.4 Performance

### 7.4.1 Single-Cut performance




Our tests show, that the runtime of the algorithm is linearly dependant on the accuracy. This is expected, as halving the error means that number of interpolation steps and therefore the number of triangles needs to be doubled.

As the nail model performance scales nearly linearly with the number of volumes which are subtracted, doubling the number of volumes only doubles the runtime. Only when the number of interpolation steps is very small the overall runtime is almost constant, as the setup time of nail model becomes a relevant factor, dominating the runtime for small sets of triangles.

### 7.4.2 Multiple-Cut performance

Often the trajectories in material removal simulations are very densely sampled, denser than the user-specified tolerance would require. Each of the sample points has a unique identifier which needs to be reported in case of a collision. Therefore, reducing the number of sampling points is not an option when the trajectory is over-sampled. As the number of triangles increases when oversampling, but the number of nail-model intersections does not, the performance bottleneck shifts from rasterizing to triangle creation. This can be seen in Figure 7.2, where the runtime of the different parts of the algorithm is compared.

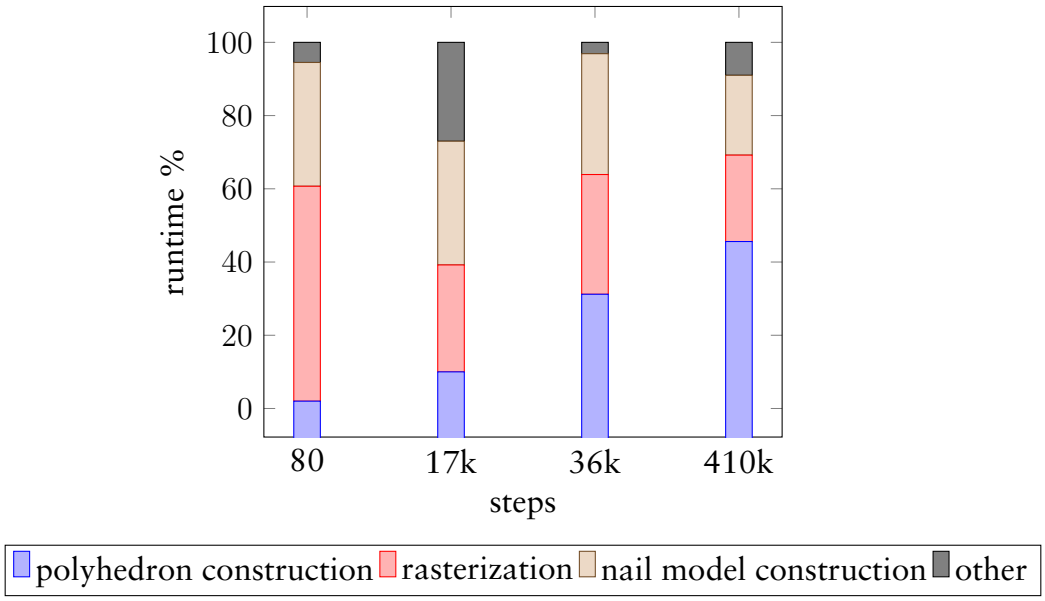



**Figure 7.2:** The same trajectory with different number of sampling steps, but the same required tolerance.

# Chapter 8

# Conclusion

The assumption that most moves are simple holds for most real-world use cases, where the tool move is densely sampled with Slerp interpolations having small rotation angles. It is usually not true for moves with big orientation changes, as we've seen in Section 7.4.1 when looking at long single cuts. The tilting cone test file shows, that the performance is relatively mediocre if many triangles lie on the rotation axis. This is expected, as it means that many triangle sweeps are complex and have to be split to stay in tolerance.

We were unfortunately unsuccessful in guaranteeing that the tolerance is never violated. However, the test files show that the specified tolerance is usually not violated. The fact that the error for ruled edges is greater the distance between edge and rotation axis is means, that triangles close to the rotation axis are oversampled. As this is the area where the error of problematic triangles is maximal, the oversampling seems to out-balance the potentially error-underestimation. The error approximation and refinement of the hull in those cases is an area worthy of further research.

A uniform step size for all triangles has been shown to yield good results. However, adaptive sampling, or different sampling of adjacent triangles (while still ensuring that the resulting volumes are gap-free) is worth further investigation, as this should reduce the number of required triangles by a lot. Especially big swept solids are much too densely sampled near the rotation axis.

The triangle rasterization might benefit from a GPU implementation of the rasterization step. To our knowledge however, to this date there are no publications that investigate how to implement the nail model data structure on a GPU. A major problem will be, that the number of intersection points per nail are not known in advance and can be very high. GPU implementations usually benefit algorithms most, which have linear access pattern, and this may be a problem because the nail model requires a huge array of dynamically

growing linear lists.

# Bibliography


[AY06] Karim Abdel-Malek and Jingzhou Yang. "Swept Volumes: Fundation, Perspectives, and Applications." In: *International Journal of Shape Modeling* 12.1 (2006), pp. 87–127.

[AY97] Karim Abdel-Malek and Harn-Jou Yeh. "Geometric representation of the swept volume using Jacobian rank-deficiency conditions". In: *Computer-Aided Design* 29.6 (1997), pp. 457–468.

[Ben+97] M. O. Benouamer et al. "Bridging the Gap between CSG and Brep via a Triple Ray Representation". In: *In Proc. Fourth ACM/Siggraph Symposium on Solid Modeling and Applications*. ACM Press, 1997, pp. 68–79.

[FMM86] Daniel Filip, Robert Magedson, and Robert Markot. "Surface algorithms using bounds on derivatives". In: *Computer Aided Geometric Design* 3.4 (1986), pp. 295–311.

[HO94] Yunching Huang and James H Oliver. "NC milling error assessment and tool path correction". In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 287–294.

[Kim+04] Young J Kim et al. "Fast swept volume approximation of complex polyhedral models". In: *Computer-Aided Design* 36.11 (2004), pp. 1013–1027.

[LKE98] In-Kwon Lee, Myung-Soo Kim, and Gershon Elber. "Polynomial/rational approximation of Minkowski sum boundary curves". In: *Graphical Models and Image Processing* 60.2 (1998), pp. 136–165.

[MS89] R. R. Martin and P. C. Stephenson. "Sweeping of Three-dimensional Objects". In: *Computer-Aided Design* 22 (1989), pp. 223–234.

[Mea80] Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[Mic14] Microsoft. *Triangle Rasterization Rules (Without Multisampling)*. 2014. URL: `http://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx`.

[SLL94] William J Schroeder, William E Lorensen, and Steve Linthicum. "Implicit modeling of swept surfaces and volumes". In: *Proceedings of the conference on Visualization'94*. IEEE Computer Society Press. 1994, pp. 40–45.

[Sho85] Ken Shoemake. "Animating rotation with quaternion curves". In: *ACM SIGGRAPH computer graphics*. Vol. 19. 3. ACM. 1985, pp. 245–254.

[WL90] John D Weld and Ming C Leu. "Geometric representation of swept volumes with application to polyhedral objects". In: *The International Journal of Robotics Research* 9.5 (1990), pp. 105–117.